Rewriting Modulo SMT and Open System Analysis

Camilo Rocha[a], José Meseguer[b], César Muñoz[c]

[a]Escuela Colombiana de Ingeniería, AK 45 205-59, Bogotá, Colombia
[b]Computer Science Department, University of Illinois, 201 Goodwin Ave, Urbana IL, USA
[c]NASA Langley Research Center, Hampton VA, USA

Abstract

This paper proposes *rewriting modulo SMT*, a new technique that combines the power of SMT solving, rewriting modulo theories, and model checking. Rewriting modulo SMT is ideally suited to model and analyze infinite-state *open systems*, i.e., systems that interact with a non-deterministic environment. Such systems exhibit both internal non-determinism, which is proper to the system, and external non-determinism, which is due to the environment. In a reflective formalism, such as rewriting logic, rewriting modulo SMT can be reduced to standard rewriting. Hence, rewriting modulo SMT naturally extends rewriting-based reachability analysis techniques, which are available for closed systems, to open systems. The proposed technique is illustrated with the formal analysis of: (i) a real-time system that is beyond the scope of timed-automata methods and (ii) automatic detection of reachability violations in a synchronous language developed to support autonomous spacecraft operations.

1. Introduction

Symbolic techniques can be used to represent possibly infinite sets of states by means of symbolic constraints. These techniques have been developed and adapted to many other verification methods such as SAT solving, Satisfiability Modulo Theories (SMT), rewriting, and model checking. A key open research issue of current symbolic techniques is extensibility. Techniques that combine different methods have been proposed, e.g., decision procedures [33, 34], unifications algorithms [7, 11], theorem provers with decision procedures [39, 1, 10], and SMT solvers in model checkers [3, 23, 32, 45, 47]. However, there is still a lack of general extensibility techniques for symbolic analysis that simultaneously combine the power of SMT solving, rewriting- and narrowing-based analysis, and model checking.

This paper proposes a new symbolic technique that seamlessly combines rewriting modulo theories, SMT solving, and model checking. For brevity, this technique is called *rewriting modulo SMT*, although it could more precisely be called *rewriting modulo SMT+B*, where B is an equational theory having a matching algorithm. It complements another symbolic technique combining narrowing modulo theories and model checking, namely narrowing-based reachability analysis [31, 8]. Neither of these two techniques subsumes the other.

Preprint submitted to Elsevier *October 1, 2014*

Rewriting modulo SMT can be applied to increase the power of equational reasoning, e.g., [26, 22, 21], but its full power, including its model checking capabilities, is better exploited when applied to concurrent open systems. Deterministic systems can be naturally specified by equational theories, but specification of concurrent, non-deterministic systems requires rewrite theories [29], i.e., triples $\mathcal{R} = (\Sigma, E, R)$ with (Σ, E) an equational theory describing system states as elements of the initial algebra $\mathcal{T}_{\Sigma/E}$, and R rewrite rules describing the system's local concurrent transitions. An *open system* is a concurrent system that interacts with an external, non-deterministic environment. When such a system is specified by a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, it has two sources of non-determinism, one internal and the other external. Internal non-determinism comes from the fact that in a given system state different instances of rules in R may be enabled. The local transitions thus enabled may lead to completely different states. What is peculiar about an open system is that it also has external, and often infinitely-branching, non-determinism due to the environment. That is, the state of an open system must include the state changes due to the environment. Technically, this means that, while a system transition in a closed system can be described by a rewrite rule $t \rightarrow t'$ with $vars(t') \subseteq vars(t)$, a transition in an open system is instead modeled by a rule of the form $t(\overrightarrow{x}) \rightarrow t'(\overrightarrow{x}, \overrightarrow{y})$, where $\overrightarrow{y}$ are fresh new variables. Therefore, a substitution for the variables $\overrightarrow{x} \uplus \overrightarrow{y}$ decomposes into two substitutions, one, say θ, for the variables $\overrightarrow{x}$ under the control of the system and another, say ρ, for the variables $\overrightarrow{y}$ under the control of the environment. In rewriting modulo SMT, such open systems are described by conditional rewrite rules of the form $t(\overrightarrow{x}) \rightarrow t'(\overrightarrow{x}, \overrightarrow{y})$ **if** ϕ, where ϕ is a constraint solvable by an SMT solver. This constraint ϕ may still allow the environment to choose an infinite number of substitutions ρ for $\overrightarrow{y}$, but can exclude choices that the environment will never make.

The non-trivial challenges of modeling and analyzing open systems can now be better explained. They include: (1) the enormous and possibly infinitary non-determinism due to the environment, which typically renders finite-state model checking impossible or unfeasible; (2) the impossibility of executing the rewrite theory $\mathcal{R} = (\Sigma, E, R)$ in the standard sense, due to the non-deterministic choice of ρ; and (3) the, in general, undecidable challenge of checking the rule's condition ϕ, since without knowing ρ, the condition $\phi\theta$ is non-ground, so that its E-satisfiability may be undecidable. As further explained in the paper, challenges (1)–(3) are all met successfully by rewriting modulo SMT because: (1) states are represented not as concrete states, i.e., ground terms, but as symbolic constrained terms $\langle t \,; \varphi \rangle$ with t a term with variables ranging in the domains handled by the SMT solver and φ an SMT-solvable formula, so that the choice of ρ is avoided; (2) rewriting modulo SMT can symbolically rewrite such pairs $\langle t \,; \varphi \rangle$ (describing possibly infinite sets of concrete states) to other pairs $\langle t' \,; \varphi' \rangle$; and (3) decidability of $\phi\theta$ (more precisely of $\varphi \wedge \phi\theta$) can be settled by invoking an SMT solver.

Rewriting modulo SMT can be integrated with model-checking by exploiting the fact that rewriting logic is reflective [15]. Hence, rewriting modulo SMT can be reduced to standard rewriting. In particular, all the techniques, algorithms, and tools available for model checking of closed systems specified as rewrite theories, such as Maude's search-based reachability analysis [14], become directly available to perform symbolic reachability analysis on systems that are now infinite-state.

The technique proposed in this paper is illustrated with the formal analysis of the

CASH scheduling protocol [13] and formal executable semantics of the Plan Execution Interchange Language (PLEXIL) [20]. The CASH protocol specifies a real-time system whose formal analysis is beyond the scope of timed-automata [2]. The language PLEXIL is a safety-critical synchronous language developed by NASA to support autonomous spacecraft operations.

This manuscript is an extended and revised version of [43]. The extension and revision include:

- Complete proofs of all results in sections 3 and 4.
- New short examples illustrating some technical definitions and results in Section 3.
- A new case study in Section 7 on automatically detecting symbolic reachability violations.

2. Preliminaries

Notation on terms, term algebras, and equational theories is used as in [6, 24].

An *order-sorted signature* Σ is a tuple $\Sigma{=}(S, \leq, F)$ with a finite poset of sorts $(S, \leq)$ and set of function symbols F. The binary relation $\equiv_{\leq}$ denotes the equivalence relation generated by $\leq$ on S and its point-wise extension to strings in S^*. The function symbols in F can be subsort-overloaded and satisfy the condition that, for $w, w' \in S^*$ and $s, s' \in S$, if $f : w \longrightarrow s$ and $f : w' \longrightarrow s'$ are in F, then $w \equiv_{\leq} w'$ implies $s \equiv_{\leq} s'$. A *top sort* in Σ is a sort $s \in S$ such that if $s' \in S$ and $s \equiv_{\leq} s'$, then $s' \leq s$. For any sort $s \in S$, the expression $[s]$ denotes the connected component of s, that is, $[s] = [s]_{\equiv_{\leq}}$.

Let $X = \{X_s\}_{s \in S}$ denote an S-indexed family of disjoint variable sets with each X_s countably infinite. The *set of terms of sort s* and the *set of ground terms of sort s* are denoted, respectively, by $T_\Sigma(X)_s$ and $T_{\Sigma,s}$; accordingly, $\mathcal{T}_\Sigma(X)$ and $\mathcal{T}_\Sigma$ denote the corresponding order-sorted Σ-term algebras. All order-sorted signatures are assumed *preregular* [24], i.e., each Σ-term t has a *least sort* $ls(t) \in S$ s.t. $t \in T_\Sigma(X)_{ls(t)}$. It is also assumed that Σ has *nonempty sorts*, i.e., $T_{\Sigma,s} \neq \emptyset$ for each $s \in S$. For $S' \subseteq S$, a term is called *S'-linear* if no variable with sort in S' occurs in it twice. The *set of variables* of t is written $vars(t)$.

A *substitution* is an S-indexed mapping $\theta : X \longrightarrow T_\Sigma(X)$ that is different from the identity only for a finite subset of X. The identity substitution is denoted by *id* and $\theta|_Y$ denotes the restriction of θ to a family of variables $Y \subseteq X$. The domain of θ, denoted $dom(\theta)$, is the subfamily of X for which $\theta(x) \neq x$, and $ran(\theta)$ denotes the family of variables introduced by the terms $\theta(x)$, such that $x \in dom(\theta)$. Substitutions extend homomorphically to terms in the natural way. A substitution θ is called *ground* iff $ran(\theta) = \emptyset$. The application of a substitution θ to a term t is denoted by $t\theta$ and the composition (in diagramatic order) of two substitutions θ_1 and θ_2 is denoted by $\theta_1\theta_2$, so that $t\theta_1\theta_2$ denotes $(t\theta_1)\theta_2$. A *context* C is a λ-term of the form $C = \lambda x_1, \ldots, x_n.c$ with $c \in T_\Sigma(X)$ and $\{x_1, \ldots, x_n\} \subseteq vars(c)$; it can be viewed as an n-ary function $C(t_1, \ldots, t_n) = c\theta$, where $\theta(x_i) = t_i$ for $1 \leq i \leq n$ and $\theta(x) = x$ otherwise.

A Σ-*equation* is an unoriented pair $t = u$ with $t \in T_\Sigma(X)_{s_t}$, $u \in T_\Sigma(X)_{s_u}$, and $s_t \equiv_{\leq} s_u$. A *conditional* Σ-*equation* is a triple $t = u$ **if** γ, with $t = u$ a Σ-equation and γ a finite conjunction of Σ-equations; a Σ-equation is called *unconditional* if γ is the empty conjunction. An *equational theory* is a tuple (Σ, E), with Σ an order-sorted signature and E a finite collection of (possibly conditional) Σ-equations. It is assumed that $T_{\Sigma,s} \neq \emptyset$ for each $s \in S$. An equational theory $\mathcal{E} = (\Sigma, E)$ induces the congruence relation $=_{\mathcal{E}}$ on $T_\Sigma(X)$ defined for $t, u \in T_\Sigma(X)$ by $t =_{\mathcal{E}} u$ iff $\mathcal{E} \vdash t = u$ by the deduction rules for order-sorted equational logic in [30]. Similarly, $=^1_{\mathcal{E}}$ denotes provable $\mathcal{E}$-equality in *one step* of deduction. The *$\mathcal{E}$-subsumption* ordering $\ll_{\mathcal{E}}$ is the binary relation on $T_\Sigma(X)$ defined for any $t, u \in T_\Sigma(X)$ by $t \ll_{\mathcal{E}} u$ iff there is a substitution $\theta : X \longrightarrow T_\Sigma(X)$ such that $t =_{\mathcal{E}} u\theta$. A set of equations E is called *collapse-free* for a subset of sorts $S' \subseteq S$ iff for any $t = u \in E$ and for any substitution $\theta : X \longrightarrow T_\Sigma(X)$ neither $t\theta$ nor $u\theta$ map to a variable having some sort $s \in S'$. The expressions $\mathcal{T}_{\mathcal{E}}(X)$ and $\mathcal{T}_{\mathcal{E}}$ (also written $\mathcal{T}_{\Sigma/E}(X)$ and $\mathcal{T}_{\Sigma/E}$) denote the quotient algebras induced by $=_{\mathcal{E}}$ on the term algebras $\mathcal{T}_\Sigma(X)$ and $\mathcal{T}_\Sigma$, respectively; $\mathcal{T}_{\Sigma/E}$ is called the *initial algebra* of (Σ, E). A theory inclusion $(\Sigma, E) \subseteq (\Sigma', E')$, with $\Sigma \subseteq \Sigma'$ and $E \subseteq E'$, is called *protecting* iff the unique Σ-homomorphism $\mathcal{T}_{\Sigma/E} \longrightarrow \mathcal{T}_{\Sigma'/E'}|_\Sigma$ to the Σ-*reduct of the initial algebra* $\mathcal{T}_{\Sigma'/E'}$ is a Σ-isomorphism, written $\mathcal{T}_{\Sigma/E} \simeq \mathcal{T}_{\Sigma'/E'}|_\Sigma$. A set of equations E is called *regular* iff $vars(t) = vars(u)$ for any equation $(t = u \textbf{ if } \gamma) \in E$.

Appropriate requirements are needed to make an equational theory $\mathcal{E}$ *admissible*, i.e., *executable* in rewriting languages such as Maude [14]. In this paper, it is assumed that the equations of $\mathcal{E}$ can be decomposed into a disjoint union $E \uplus B$, with B a collection of regular and linear structural axioms (such as associativity, and/or commutativity, and/or identity) for which there exists a *matching algorithm modulo B* producing a finite number of B-matching solutions, or failing otherwise. Furthermore, it is assumed that the equations E can be oriented into a set of (possibly conditional) strongly deterministic [35], sort-decreasing, operationally terminating, confluent, and strictly B-coherent [19] conditional rewrite rules $\overrightarrow{E}$ modulo B. The conditional rewrite system $\overrightarrow{E}$ is *sort decreasing* modulo B iff for each $(t \to u \textbf{ if } \gamma) \in \overrightarrow{E}$ and substitution θ, $ls(t\theta) \geq ls(u\theta)$ if $(\Sigma, B, \overrightarrow{E}) \vdash \gamma\theta$. The system $\overrightarrow{E}$ is *operationally terminating* modulo B iff there is no infinite well-formed proof tree in $(\Sigma, B, \overrightarrow{E})$. Furthermore, $\overrightarrow{E}$ is *confluent* modulo B iff for all $t, t_1, t_2 \in T_\Sigma(X)$, if $t \to^*_{E/B} t_1$ and $t \to^*_{E/B} t_2$, then there is $u \in T_\Sigma(X)$ such that $t_1 \to^*_{E/B} u$ and $t_2 \to^*_{E/B} u$. The term $t\downarrow_{E/B} \in T_\Sigma(X)$ denotes the *E-canonical form* of t modulo B so that $t \to^*_{E/B} t\downarrow_{E/B}$ and $t\downarrow_{E/B}$ cannot be further reduced by $\to_{E/B}$. Under the above assumptions $t\downarrow_{E/B}$ is unique up to B-equality.

A Σ-*rule* is a triple $l \to r \textbf{ if } \phi$, with $l, r \in T_\Sigma(X)_s$, for some sort $s \in S$, and $\phi = \bigwedge_{i \in I} t_i = u_i$ a finite conjunction of Σ-equations. A *rewrite theory* is a tuple $\mathcal{R} = (\Sigma, E, R)$ with (Σ, E) an order-sorted equational theory and R a finite set of Σ-rules. The rewrite theory $\mathcal{R}$ induces a rewrite relation $\to_{\mathcal{R}}$ on $T_\Sigma(X)$ defined for every $t, u \in T_\Sigma(X)$ by $t \to_{\mathcal{R}} u$ iff there is a rule $(l \to r \textbf{ if } \phi) \in R$ and a substitution $\theta : X \longrightarrow T_\Sigma(X)$ satisfying $t =_E l\theta$, $u =_E r\theta$, and $E \vdash \phi\theta$. The relation $\to_{\mathcal{R}}$ is undecidable in general, unless conditions such as coherence [46] are given. A key point of this paper is to make such a relation decidable when E decomposes as $\mathcal{E}_0 \uplus B_1$, where $\mathcal{E}_0$ is a built-in theory for which formula satisfiability is decidable and B_1 has a matching algorithm. A *topmost rewrite theory* is a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, such that for some top sort *State*, no operator in Σ has *State* as argument sort and each rule $l \to r \textbf{ if } \phi \in R$ satisfies $l, r \in T_\Sigma(X)_{State}$ and $l \notin X$.

3. Rewriting Modulo a Built-in Subtheory

This section introduces the concept of rewriting modulo a built-in equational subtheory and presents its main properties.

Definition 1 (Signature with Built-ins)**.** *An order-sorted signature $\Sigma = (S, \leq, F)$ is a signature with* built-in subsignature $\Sigma_0 \subseteq \Sigma$ *iff $\Sigma_0 = (S_0, F_0)$ is many-sorted, S_0 is a set of minimal elements in $(S, \leq)$, and if $f : w \longrightarrow s \in F_1$, then $s \notin S_0$ and f has no other typing in F_0, where $F_1 = F \backslash F_0$.*

The notion of built-in subsignature in an order-sorted signature Σ is modeled by a many-sorted signature Σ_0 defining the built-in terms $T_{\Sigma_0}(X_0)$. The restriction imposed on the sorts and the function symbols in Σ w.r.t. Σ_0 provides a clear syntactic distinction between built-in terms (the only ones with built-in sorts) and all other terms.

Example 1. Consider the following order-sorted signature in the syntax of Maude:

```
sorts Nat AttributeName Attribute AttrSet .
op 0 : -> Nat .
op s_ : Nat -> Nat .
ops maxBudget timeToDeadline : -> AttributeName .
op _|->_ : AttributeName Nat -> Attribute .
op mt : -> AttrSet .
op _,_ : AttrSet AttrSet -> AttrSet [assoc comm id: mt] .
```

This signature models a multiset of named attributes similar to the ones that are currently employed in algebraic object-like specifications. Sort `Nat` specifies natural numbers in Peano notation and sort `AttributeName` attribute names. A named attribute in `Attribute` is term `AN |-> N` with `AN` an attribute name and `N` a natural number. Sort `AttrSet` specifies multisets of named attributes with multiset union denoted by '`,`' and with identity '`id`'. The following is a term in `AttrSet` denoting that `maxbudget` is 2 and `timeToDeadline` is 1:

```
maxbudget |-> s(s(0)), timeToDeadline |-> s(0)
```

In this case, the many-sorted signature $\Sigma_0 = (\{\texttt{Nat}\}, \{\texttt{0}, \texttt{s}\})$ is a built-in subsignature of the order-sorted signature. Finally, F_1 includes all function symbols in the signature except for those in the set $\{\texttt{0}, \texttt{s}\}$.

If $\Sigma \supseteq \Sigma_0$ is a signature with built-ins, then an *abstraction of built-ins* for t is a context $\lambda x_1 \cdots x_n.t^\circ$ such that $t^\circ \in T_{\Sigma_1}(X)$ and $\{x_1, \ldots, x_n\} = vars(t^\circ) \cap X_0$, where $\Sigma_1 = (S, \leq, F_1)$ and $X_0 = \{X_s\}_{s \in S_0}$. Lemma 1 shows that such an abstraction can be chosen so as to provide a canonical decomposition of t with useful properties.

Lemma 1. *Let Σ be a signature with built-in subsignature $\Sigma_0 = (S_0, F_0)$. For each $t \in T_\Sigma(X)$, there exist an abstraction of built-ins $\lambda x_1 \cdots x_n.t^\circ$ for t and a substitution $\theta^\circ : X_0 \longrightarrow T_{\Sigma_0}(X_0)$ such that (i) $t = t^\circ\theta^\circ$ and (ii) $dom(\theta^\circ) = \{x_1, \ldots, x_n\}$ are pairwise distinct and disjoint from $vars(t)$; moreover, (iii) t° can always be selected to be S_0-linear and with $\{x_1, \ldots, x_n\}$ disjoint from an arbitrarily chosen finite subset Y of X_0.*

Proof. By induction on the structure of t. □

In the rest of the paper, for any $t \in T_\Sigma(X)$ and $Y \subseteq X_0$ finite, the expression $abstract_{\Sigma_1}(t, Y)$ denotes the choice of a triple $\langle \lambda x_1 \cdots x_n.t^\circ ; \theta^\circ ; \phi^\circ \rangle$ such that the context $\lambda x_1 \cdots x_n.t^\circ$ and the substitution θ° satisfy the properties (i)–(iii) in Lemma 1 and $\phi^\circ = \bigwedge_{i=1}^{n} (x_i = \theta^\circ(x_i))$.

Example 2. Let t be the term

```
maxbudget |-> s(s(N1:Nat)), timeToDeadline |-> s(N2:Nat), AttS:AttrSet
```

in the signature of Example 1, where `N1,N2` are variables of sort `Nat` and `AttS` is a variable of sort `AttrSet`. Consider the term t°

```
maxbudget |-> N3:Nat, timeToDeadline |-> N4:Nat, AttS:AttrSet
```

and the substitution θ° defined by $\theta^\circ(\texttt{N3}) = \texttt{s(s(N1))}$, $\theta^\circ(\texttt{N4}) = \texttt{s(N2)}$, and $\theta^\circ(x) = x$ otherwise. Then the context $\lambda\texttt{N3},\texttt{N4}.t^\circ$ is an abstraction of built-ins for t and θ° satisfies properties (i)–(iii) in Lemma 1. Moreover, for any set Y not containing variables `N3` or `N4`, t° and θ° satisfy $abstract_{\Sigma_1}(t, Y) = \langle \lambda x_1 \cdots x_n.t^\circ ; \theta^\circ ; \phi^\circ \rangle$ with ϕ° denoting the constraint

$$\texttt{N3} = \texttt{s(s(N1))} \ \wedge \ \texttt{N4} = \texttt{s(N2)}.$$

Under certain restrictions on axioms, matching a Σ-term t to a Σ-term u can be decomposed modularly into Σ_1-matching of the corresponding λ-abstraction and Σ_0-matching of the built-in subterms. This is described in Lemma 2, with the help of Corollary 1.

Corollary 1. *Let $\Sigma = (S, \leq, F)$ be a signature with built-in subsignature $\Sigma_0 = (S_0, F_0)$. Let B_0 be a set of Σ_0-axioms and B_1 a set of Σ_1-axioms. For B_0 and B_1 regular, linear, collapse free for any sort in S_0, and sort-preserving, and $t \in T_\Sigma(X_0)$:*

(a) if $t \in T_{\Sigma_0}(X_0)$ and $t =^1_{B_1} t'$, then $t = t'$;

(b) if $t \in T_{\Sigma_1}(X_0)$ and $t =^1_{B_0} t'$, then $t = t'$;

(c) if $t \in T_{\Sigma_1}(X_0)$ and $t =^1_{B_1} t'$, then $vars(t) = vars(t')$ and t is linear iff t' is so;

Proof.

(a) Axioms B_1 do not mention any function symbol in F_0. Therefore, the equation in B_0 can only apply to variables in X_0. But B_1 is collapse-free for any sort in S_0. Therefore, no B_1 equation can be applied to t, forcing $t = t'$.

(b) Same argument as (a).

(c) Consequence of B_1 being regular and linear.

□

Lemma 2. *Let $\Sigma = (S, \leq, F)$ be a signature with built-in subsignature $\Sigma_0 = (S_0, F_0)$. Let B_0 be a set of Σ_0-axioms and B_1 a set of Σ_1-axioms. For B_0 and B_1 regular, linear, collapse free for any sort in S_0, and sort-preserving, if $t \in T_{\Sigma_1}(X_0)$ is linear with $vars(t) = \{x_1, \ldots, x_n\}$, then for each $\theta : X_0 \longrightarrow T_{\Sigma_0}(X_0)$:*

(a) *if* $t\theta =^1_{B_0} t'$*, then there exist* $x \in \{x_1, \ldots, x_n\}$ *and* $w \in T_{\Sigma_0}(X_0)$ *such that* $\theta(x) =^1_{B_0} w$ *and* $t' = t\theta'$*, with* $\theta'(x) = w$ *and* $\theta'(y) = \theta(y)$ *otherwise;*

(b) *if* $t\theta =^1_{B_1} t'$*, then there exists* $v \in T_{\Sigma_1}(X_0)$ *such that* $t =^1_{B_1} v$ *and* $t' = v\theta$*; and*

(c) *if* $t\theta =_{B_0 \uplus B_1} t'$*, then there exist* $v \in T_{\Sigma_1}(X_0)$ *and* $\theta' : X_0 \longrightarrow T_{\Sigma_0}(X_0)$ *such that* $t' = v\theta'$*,* $t =_{B_1} v$*, and* $\theta =_{B_0} \theta'$ *(i.e.,* $\theta(x) =_{B_0} \theta'(x)$ *for each* $x \in X_0$*).*

Proof. (a) It follows from Corollary 1 part (b) that B_0 can only be applied on some built-in subterm $\theta(x)$ of $t\theta$, for some $x \in dom(\theta)$. That is, there is $w \in T_{\Sigma_0}(X_0)$ such that $\theta(x) =^1_{B_0} w$ and, since t is linear, $t' = t\theta'$, where $\theta'(x) = w$ and $\theta'(x) = \theta(x)$ otherwise.

(b) It follows from Corollary 1 part (c) that equational deduction with B_1 can only permute the built-in variables in t and it does not equate built-in subterms such as the ones in $ran(\theta)$. Hence, by Corollary 1 part (c), there exists a linear $v \in T_{\Sigma_1}(X_0)$ such that $t =^1_{B_1} v$ and $t' = v\theta$.

(c) Follows by induction on the proof's length in $B_0 \uplus B_1$.

□

Definition 2 (Rewriting Modulo a Built-in Subtheory)**.** *A* rewrite theory modulo the built-in subtheory $\mathcal{E}_0$ *is a topmost rewrite theory* $\mathcal{R} = (\Sigma, E, R)$ *with:*

(a) $\Sigma = (S, \leq, F)$ *a signature with built-in subsignature* $\Sigma_0 = (S_0, F_0)$ *and top sort* $State \in S$*;*

(b) $E = E_0 \uplus B_0 \uplus B_1$*, where* E_0 *is a set of* Σ_0*-equations,* B_0 *(resp.,* B_1*) are* Σ_0*-axioms (resp.,* Σ_1*-axioms) satisfying the conditions in Lemma 2,* $\mathcal{E}_0 = (\Sigma_0, E_0 \uplus B_0)$ *and* $\mathcal{E} = (\Sigma, E)$ *are admissible, and the theory inclusion* $\mathcal{E}_0 \subseteq \mathcal{E}$ *is protecting;*

(c) R *is a set of* rewrite rules *of the form* $l(\overrightarrow{x_1}, \overrightarrow{y}) \to r(\overrightarrow{x_2}, \overrightarrow{y})$ ***if*** $\phi(\overrightarrow{x_3})$ *such that* $l, r \in T_\Sigma(X)_{State}$*,* l *is* $(S \setminus S_0)$*-linear,* $\overrightarrow{x_i}{:}\overrightarrow{s_i}$ *with* $\overrightarrow{s_i} \in S_0^*$*, for* $i \in \{1, 2, 3\}$*,* $\overrightarrow{y}:\overrightarrow{s}$ *with* $\overrightarrow{s} \in (S \setminus S_0)^*$*, and* $\phi \in QF_{\Sigma_0}(X_0)$*, where* $QF_{\Sigma_0}(X_0)$ *denotes the set of quantifier-free* Σ_0*-formulas with variables in* X_0*.*

Note that no assumption is made on the relationship between the built-in variables $\overrightarrow{x_1}$ in the left-hand side, $\overrightarrow{x_2}$ in the right-hand side, and $\overrightarrow{x_3}$ in the condition ϕ of a rewrite rule. This freedom is key for specifying open systems with a rewrite theory because, for instance, $\overrightarrow{x_2}$ can have more variables than $\overrightarrow{x_1}$. On the other hand, due to the presence of conditions ϕ in the rules of $\mathcal{R}$ that are general quantifier-free formulas, as opposed to a conjunction of atoms, properly speaking $\mathcal{R}$ is more general than a standard rewrite theory as defined in Section 2.

The binary rewrite relation induced by a rewrite theory $\mathcal{R}$ modulo $\mathcal{E}_0$ on $T_{\Sigma,State}$ is called the *ground rewrite relation* of $\mathcal{R}$.

Definition 3 (Ground Rewrite Relation)**.** *Let* $\mathcal{R} = (\Sigma, E, R)$ *be a rewrite theory modulo* $\mathcal{E}_0$*. The relation* $\to_{\mathcal{R}}$ *induced by* $\mathcal{R}$ *on* $T_{\Sigma,State}$ *is defined for* $t, u \in T_{\Sigma,State}$ *by* $t \to_{\mathcal{R}} u$ *iff there is a rule* $l \to r$ ***if*** ϕ *in* R *and a ground substitution* $\sigma : X \longrightarrow T_\Sigma$ *such that (a)* $t =_E l\sigma$*,* $u =_E r\sigma$*, and (b)* $\mathcal{T}_{\mathcal{E}_0} \models \phi\sigma$*.*

The ground rewrite relation $\to_{\mathcal{R}}$ is the topmost rewrite relation induced by R modulo E on $T_{\Sigma,State}$. This relation is defined even when a rule in R has extra variables in its right-hand side: the rule is then non-deterministic and such extra variables can be arbitrarily instantiated, provided that the corresponding instantiation of ϕ holds. Also, note that non-built-in variables can occur in l, but $\phi\sigma$ is a *variable-free formula* in $QF_{\Sigma_0}(\emptyset)$, so that either $\mathcal{T}_{\mathcal{E}_0} \models \phi\sigma$ or $\mathcal{T}_{\mathcal{E}_0} \not\models \phi\sigma$.

A rewrite theory $\mathcal{R}$ modulo $\mathcal{E}_0$ always has a canonical representation in which all left-hand sides of rules are S_0-linear Σ_1-terms.

Definition 4 (Normal Form of a Rewrite Theory Modulo $\mathcal{E}_0$)**.** *Let* $\mathcal{R} = (\Sigma, E, R)$ *be a rewrite theory modulo* $\mathcal{E}_0$*. Its* normal form $\mathcal{R}^\circ = (\Sigma, E, R^\circ)$ *has rules:*

$$R^\circ = \{l^\circ \to r \textbf{ if } \phi \wedge \phi^\circ \mid (\exists l \to r \textbf{ if } \phi \in R)\langle \lambda \overrightarrow{x}.l^\circ\,;\theta^\circ\,;\phi^\circ\rangle = abstract_\Sigma(l, vars(\{l, r, \phi\}))\}.$$

Lemma 3 (Invariance of Ground Rewriting under Normalization)**.** *Let* $\mathcal{R} = (\Sigma, E, R)$ *be a rewrite theory modulo* $\mathcal{E}_0$*. Then* $\to_{\mathcal{R}} = \to_{\mathcal{R}^\circ}$*.*

Proof. It is shown that $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}^\circ}$ and $\to_{\mathcal{R}^\circ} \subseteq \to_{\mathcal{R}}$.

($\subseteq$) Let $t, u \in T_{\Sigma,State}$. If $t \to_{\mathcal{R}} u$, then there is a rule $(l \to r$ **if** $\phi) \in R$ and a ground substitution $\sigma : X \longrightarrow T_\Sigma$ such that $t =_E l\sigma$, $u =_E r\sigma$, and $\mathcal{T}_{\mathcal{E}_0} \models \phi\sigma$. It suffices to prove $t \to_{\mathcal{R}^\circ} u$ with witnesses $(l^\circ \to r$ **if** $\phi \wedge \phi^\circ) \in R^\circ$ and $\rho = \theta^\circ\sigma$. Note that $t =_E l\sigma = l^\circ\theta^\circ\sigma = l^\circ\rho$. For $\mathcal{T}_{\mathcal{E}_0} \models (\phi \wedge \phi^\circ)\rho$ first note that $\mathcal{T}_{\mathcal{E}_0} \models \phi\rho$ since $\phi\rho = \phi\theta^\circ\sigma = \phi\sigma$ (because $vars(\phi) \cap dom(\theta^\circ) = \emptyset$) and $\mathcal{T}_{\mathcal{E}_0} \models \phi\sigma$ by assumption. For $\mathcal{T}_{\mathcal{E}_0} \models \phi^\circ\rho$ notice that $\theta^\circ\theta^\circ = \theta^\circ$ because $ran(\theta^\circ) \cap dom(\theta^\circ) = \emptyset$, and then:

$$\begin{aligned}\phi^\circ\rho &= \left(\bigwedge_{i=1}^{n} x_i = \theta^\circ(x_i)\right)\rho = \bigwedge_{i=1}^{n} x_i\rho = \theta^\circ(x_i)\rho = \bigwedge_{i=1}^{n} \theta^\circ(x_i)\sigma = \theta^\circ(x_i)\theta^\circ\sigma \\ &= \bigwedge_{i=1}^{n} \theta^\circ(x_i)\sigma = \theta^\circ(x_i)\sigma = \top.\end{aligned}$$

Hence, $t \to_{\mathcal{R}^\circ} u$.

($\supseteq$) Let $t, u \in T_{\Sigma,State}$. If $t \to_{\mathcal{R}}^\circ u$, then there is a rule $(l \to r$ **if** $\phi) \in R$ and a ground substitution $\sigma : X \longrightarrow T_\Sigma$ such that $t =_E l^\circ\sigma$, $u =_E r\sigma$, and $\mathcal{T}_{\mathcal{E}_0} \models (\phi \wedge \phi^\circ)\sigma$. It suffices to prove $t \to_{\mathcal{R}} u$ with witness $(l \to r$ **if** $\phi) \in R$. Let $\langle \lambda x_1 \cdots x_n.l^\circ\,;\theta^\circ\,;\phi^\circ\rangle$ be the abstraction of built-ins for l. Substitution σ can be decomposed into substitutions $\theta : X_0 \longrightarrow T_{\Sigma_0}(X_0)$ and $\rho : X \longrightarrow T_\Sigma$, with $\theta(x) = \sigma(x)$ if $x \in \{x_1, \ldots, x_n\}$ and $\theta(x) = x$ otherwise, such that $\sigma = \theta\rho$. From $\mathcal{T}_{\mathcal{E}_0} \models (\phi \wedge \phi^\circ)\sigma$ it follows that $\mathcal{T}_{\mathcal{E}_0} \models \phi\sigma$, i.e., $\mathcal{T}_{\mathcal{E}_0} \models \phi\rho$ because $vars(\phi) \cap dom(\theta) = \emptyset$. Also, it follows that $\mathcal{T}_{\mathcal{E}_0} \models \bigwedge_{i=1}^{n} \theta(x_i)\rho = \theta^\circ(x_i)\rho$, which implies that:

$$t =_E l^\circ\sigma = l^\circ\theta\rho =_{E_0 \uplus B_0} l^\circ\theta^\circ\rho = l\rho.$$

Hence, $t \to_{\mathcal{R}} u$.

$\square$

By the properties of the axioms in a rewrite theory modulo built-ins $\mathcal{R} = (\Sigma, E_0 \uplus B_0 \uplus B_1)$ (see Definition 2), B_1-matching a term $t \in T_\Sigma(X_0)$ to a left-hand side l° of a rule in R° provides a complete unifiability algorithm for ground B_1-unification of t and l°.

Lemma 4 (Matching Lemma)**.** *Let $\mathcal{R} = (\Sigma, E_0 \uplus B_0 \uplus B_1, R)$ be a rewrite theory modulo $\mathcal{E}_0$. For $t \in T_\Sigma(X_0)_{State}$ and l° a left-hand side of a rule in R° with $vars(t) \cap vars(l^\circ) = \emptyset$,*

$$t \ll_{B_1} l^\circ \quad \textit{iff} \quad GU_{B_1}(t = l^\circ) \neq \emptyset$$

where $GU_{B_1}(t = l^\circ) = \{\sigma : X \longrightarrow T_\Sigma \mid t\sigma =_{B_1} l^\circ\sigma\}$.

Proof.

($\Longrightarrow$) If $t \ll_{B_1} l^\circ$, then $t =_{B_1} l^\circ\theta$ for some $\theta : X \longrightarrow T_\Sigma(X)$. Let $\rho : X \longrightarrow T_\Sigma$ be any ground substitution, which exists because Σ has nonempty sorts. Then $\theta\rho \in GU_{B_1}(t = l^\circ)$.

($\Longleftarrow$) Let $\sigma \in GU_{B_1}(t = l^\circ)$ with $l \to r$ **if** $\phi \in R$. Let $vars(l^\circ) \cap X_0 = \{x_1, \ldots, x_n\}$ and $X_1 = X \setminus X_0$. Note that there are substitutions

$$\begin{aligned} &\alpha : vars(l^\circ) \cap X_1 \longrightarrow T_{\Sigma_1}(X_0) \\ &\rho : X \setminus dom(\alpha) \longrightarrow T_\Sigma \end{aligned}$$

satisfying $\sigma = \alpha\rho$ and such that $(l^\circ\alpha) \in T_{\Sigma_1}(X_0)$ is linear and

$$ran(l^\circ\alpha) \cap (vars(t, l^\circ)) = \emptyset.$$

Let $ran(\alpha) = \{y_1, \ldots, y_m\}$. Therefore, by Lemma 2, there exists $u \in T_{\Sigma_1}(X_0)$ such that $u =_{B_1} l^\circ\alpha$, u is linear, and $vars(u) = vars(l^\circ\alpha) = x_1, \ldots, x_n, y_1, \ldots, y_m$, and $u\rho = t$. Moreover, t can be written as $u(t_1, \ldots, t_n, t_{n+1}, \ldots, t_{n+m})$ with $t_i \in T_{\Sigma_0}(X_0)$. Define $\theta : X_0 \longrightarrow T_{\Sigma_0}(X_0)$ by $\theta(x) = t_i$ if $x \in \{x_1, \ldots, x_n\}$, $\theta(x) = t_{i+n}$ if $x \in \{y_1, \ldots, y_m\}$, and $\theta(x) = x$ otherwise. Then we have:

$$\begin{aligned} t &= u(t_1, \ldots, t_n, t_{n+1}, \ldots, t_{m+n}) \\ &= u(x_1, \ldots, x_n, y_1, \ldots, y_m)\theta \\ &=_{B_1} l^\circ\alpha\theta. \end{aligned}$$

Therefore, $t \ll_{B_1} l^\circ$.

□

4. Symbolic Rewriting Modulo a Built-in Subtheory

This section explains how a rewrite theory $\mathcal{R}$ modulo $\mathcal{E}_0$ defines a symbolic rewrite relation on terms in $T_{\Sigma_0}(X_0)_{State}$ constrained by formulas in $QF_{\Sigma_0}(X_0)$. The key idea is that, when $\mathcal{E}_0$ is a decidable theory, transitions on the symbolic terms can be performed by rewriting modulo B_1, and satisfiability of the formulas can be handled by an SMT decision procedure. This approach provides an efficiently executable symbolic method called *rewriting modulo SMT* that is sound and complete with respect to the ground rewrite relation of Definition 3 and yields a complete symbolic reachability analysis method.

Definition 5 (Constrained Terms and their Denotation)**.** *Let* $\mathcal{R} = (\Sigma, E, R)$ *be a rewrite theory modulo* $\mathcal{E}_0$*. A* constrained term *is a pair* $\langle t\,;\varphi\rangle$ *in* $T_\Sigma(X_0)_{State} \times QF_{\Sigma_0}(X_0)$*. Its* denotation $[\![t]\!]_\varphi$ *is defined as* $[\![t]\!]_\varphi = \{t' {\in} T_{\Sigma,State} \mid (\exists\sigma : X_0 {\longrightarrow} T_{\Sigma_0})\ t'{=}t\sigma \wedge \mathcal{T}_{\mathcal{E}_0} \models \varphi\sigma\}$*.*

The domain of σ in Definition 5 ranges over all built-in variables X_0 and consequently $[\![t]\!]_\varphi \subseteq T_{\Sigma,State}$ for any $t \in T_\Sigma(X_0)_{State}$, even if $vars(t) \not\subseteq vars(\varphi)$. Intuitively, $[\![t]\!]_\varphi$ denotes the set of all ground states that are instances of t and satisfy φ.

Before introducing the symbolic rewrite relation on constrained terms induced by a rewrite theory $\mathcal{R}$ modulo $\mathcal{E}_0$, auxiliary notation for variable renaming is required. In the rest of the paper, the expression *fresh-vars*(Y), for $Y \subseteq X$ finite, represents the choice of a variable renaming $\zeta : X \longrightarrow X$ satisfying $Y \cap ran(\zeta) = \emptyset$.

Definition 6 (Symbolic Rewrite Relation)**.** *Let* $\mathcal{R} = (\Sigma, E, R)$ *be a rewrite theory modulo built-ins* $\mathcal{E}_0$*. The* symbolic rewrite relation $\rightsquigarrow_\mathcal{R}$ *induced by* $\mathcal{R}$ *on* $T_\Sigma(X_0)_{State} \times QF_{\Sigma_0}(X_0)$ *is defined for* $t, u \in T_\Sigma(X_0)_{State}$ *and* $\varphi, \varphi' \in QF_{\Sigma_0}(X_0)$ *by* $\langle t\,;\varphi\rangle \rightsquigarrow_\mathcal{R} \langle u\,;\varphi'\rangle$ *iff there is a rule* $l \to r$ ***if*** ϕ *in* R *and a substitution* $\theta : X \longrightarrow T_\Sigma(X)$ *such that (a)* $t =_E l\zeta\theta$ *and* $u = r\zeta\theta$*, (b)* $\mathcal{E}_0 \vdash (\varphi' \Leftrightarrow \varphi \wedge \phi\zeta\theta)$*, and (c)* φ' *is* $\mathcal{T}_{\mathcal{E}_0}$*-satisfiable, where* $\zeta = fresh\text{-}vars(vars(t,\varphi))$*.*

The symbolic relation $\rightsquigarrow_\mathcal{R}$ on constrained terms is defined as a topmost rewrite relation induced by R modulo E on $T_\Sigma(X_0)$ with extra bookkeeping of constraints. Note that φ' in $\langle t\,;\varphi\rangle \rightsquigarrow_\mathcal{R} \langle u\,;\varphi'\rangle$, when witnessed by $l \to r$ **if** ϕ and θ, is *semantically equivalent* to $\varphi \wedge \phi\zeta\theta$, in contrast to being *syntactically* equal. This extra freedom allows for simplification of constraints if desired. Also, such a constraint φ' is satisfiable in $\mathcal{T}_{\mathcal{E}_0}$, implying that φ and $\phi\theta$ are both satisfiable in $\mathcal{T}_{\mathcal{E}_0}$, and therefore $[\![t]\!]_\varphi \neq \emptyset \neq [\![u]\!]_{\varphi'}$. Note that, up to the choice of the semantically equivalent φ' for which a fixed strategy is assumed, the symbolic relation $\rightsquigarrow_\mathcal{R}$ is "deterministic", in the sense of being determined by the rule and the substitution $\zeta\theta$, because the renaming of variables in the rules is fixed by *fresh-vars*. This is key when executing $\rightsquigarrow_\mathcal{R}$, as explained in Section 5.

The important question to ask is whether this symbolic relation soundly and completely simulates its ground counterpart. The rest of this section affirmatively answers this question in the case of *normalized* rewrite theories modulo built-ins. Thanks to Lemma 3, the conclusion is therefore that $\rightsquigarrow_{\mathcal{R}^\circ}$ soundly and completely simulates $\to_\mathcal{R}$ for any rewrite theory $\mathcal{R}$ modulo built-ins $\mathcal{E}_0$.

The soundness of $\rightsquigarrow_{\mathcal{R}^\circ}$ w.r.t. $\to_{\mathcal{R}^\circ}$ is stated in Theorem 1.

Theorem 1 (Soundness)**.** *Let* $\mathcal{R} = (\Sigma, E, R)$ *be a rewrite theory modulo built-ins* $\mathcal{E}_0$*,* $t, u \in T_\Sigma(X_0)_{State}$*, and* $\varphi, \varphi' \in QF_{\Sigma_0}(X_0)$*. If* $\langle t\,;\varphi\rangle \rightsquigarrow_{\mathcal{R}^\circ} \langle u\,;\varphi'\rangle$*, then* $t\rho \to_{\mathcal{R}^\circ} u\rho$ *for all* $\rho : X_0 \longrightarrow T_{\Sigma_0}$ *satisfying* $\mathcal{T}_{\mathcal{E}_0} \models \varphi'\rho$*.*

Proof. Let $\rho : X_0 \longrightarrow T_{\Sigma_0}$ satisfy $\mathcal{T}_{\mathcal{E}_0} \models \varphi'\rho$. The goal is to show that $t\rho \to_{\mathcal{R}^\circ} u\rho$. Let $l^\circ \to r$ **if** $\phi \in R^\circ$ and $\theta : X_0 \longrightarrow T_{\Sigma_0}(X_0)$ witness $\langle t\,;\varphi\rangle \rightsquigarrow_{\mathcal{R}^\circ} \langle u\,;\varphi'\rangle$. Then $t =_E l^\circ\zeta\theta$, $u =_E r\zeta\theta$, $\mathcal{E}_0 \vdash (\varphi' \Leftrightarrow \varphi \wedge \phi\zeta\theta)$, and φ' is $\mathcal{T}_{\mathcal{E}_0}$-satisfiable. Without loss of generality assume $dom(\theta) = vars(l^\circ\zeta)$ and $\theta|_{vars(t,\varphi)} = id$, and let $\sigma = \zeta\theta\rho$. Then note that $t\rho =_E (l^\circ\zeta\theta)\rho = l^\circ\zeta\theta\rho = l^\circ\sigma$ and $u\rho =_E (r\zeta\theta)\rho = r\zeta\theta\rho = r\sigma$. Moreover, $\mathcal{T}_{\mathcal{E}_0} \models (\varphi' \Leftrightarrow \varphi \wedge \phi\zeta\theta)$ and $\mathcal{T}_{\mathcal{E}_0} \models \varphi'\rho$ imply $\mathcal{T}_{\mathcal{E}_0} \models \phi\zeta\theta\rho$, i.e., $\mathcal{T}_{\mathcal{E}_0} \models \phi\sigma$. Therefore, $t\rho \rightarrow_{\mathcal{R}^\circ} u\rho$, as desired. □

The completeness of $\rightsquigarrow_{\mathcal{R}^\circ}$ w.r.t. $\rightarrow_{\mathcal{R}^\circ}$ is stated in Theorem 2. Intuitively, completeness states that a symbolic relation yields an over-approximation of its ground rewriting counterpart.

Theorem 2 (Completeness)**.** *Let $\mathcal{R} = (\Sigma, E, R)$ be a rewrite theory modulo built-ins $\mathcal{E}_0$, $t \in T_\Sigma(X_0)_{State}$, $u' \in T_{\Sigma,State}$, and $\varphi \in QF_{\Sigma_0}(X_0)$. For any $\rho : X_0 \longrightarrow T_{\Sigma_0}$ such that $t\rho \in [\![t]\!]_\varphi$ and $t\rho \rightarrow_{\mathcal{R}^\circ} u'$, there exist $u \in T_\Sigma(X_0)_{State}$ and $\varphi' \in QF_{\Sigma_0}(X_0)$ such that $\langle t\,;\varphi\rangle \rightsquigarrow_{\mathcal{R}^\circ} \langle u\,;\varphi'\rangle$ and $u' \in [\![u]\!]_{\varphi'}$.*

Proof. By the assumptions there is a rule $(l^\circ \to r \textbf{ if } \phi) \in R^\circ$ and a ground substitution $\sigma : X \longrightarrow T_\Sigma$ satisfying $t\rho =_E l^\circ\sigma$, $u' =_E r\sigma$, and $\mathcal{T}_{\mathcal{E}_0} \models \phi\sigma$. Without loss of generality assume $vars(t,\varphi) \cap vars(l^\circ, r, \phi)) = \emptyset$; otherwise l, r, ϕ can be renamed by means of *fresh-vars*. Furthermore, since $vars(t,\varphi) \cap vars(l^\circ,\phi)) = \emptyset$, $\sigma = \rho$ can be assumed. The goal is to show the existence of $u \in T_\Sigma(X)_{State}$ and $\varphi' \in QF_{\Sigma_0}(X_0)$ such that (i) $\langle t\,;\varphi\rangle \rightsquigarrow_{\mathcal{R}^\circ} \langle u\,;\varphi'\rangle$ and (ii) $u' \in [\![u]\!]_{\varphi'}$. Since l° is linear and built-in subterms are variables, by Lemma 2 there exists $\alpha : X \longrightarrow T_\Sigma$ satisfying $t\alpha =_{B_1} l^\circ\alpha$. Hence $GU_{B_1}(t = l^\circ) \neq \emptyset$ and, by Lemma 4, there exists $\theta' : X \longrightarrow T_\Sigma(X)$ satisfying $t =_{B_1} l^\circ\theta'$ and a fortiori $t =_{E_0 \uplus B_0 \uplus B_1} l^\circ\theta'$. Let $\theta : X \longrightarrow T_\Sigma(X)$ be defined by $\theta(x) = \theta'(x)$ if $x \in vars(l)$ and $\theta(x) = \rho(x)$ otherwise. Note that $\theta|_{vars(l)}\rho =_{E_0 \uplus B_0} \rho|_{vars(l)}$. Define $u = r\theta$ and $\varphi' = \varphi \wedge \phi\theta$, and then for (i) and (ii) above:

(i) It suffices to prove that $\mathcal{T}_{\mathcal{E}_0} \models \varphi'\rho$, i.e., $\mathcal{T}_{\mathcal{E}_0} \models (\varphi \wedge \phi\theta)\rho$. By assumption $\mathcal{T}_{\mathcal{E}_0} \models \varphi\rho$ and $\mathcal{T}_{\mathcal{E}_0} \models \phi\rho$. Notice that:

$$\phi\theta\rho = (\phi\theta|_{vars(l)})\rho =_{E_0 \uplus B_0} (\phi\rho)\rho = \phi\rho.$$

Hence $\mathcal{T}_{\mathcal{E}_0} \models \phi\theta\rho$.

(ii) By assumption $u' =_{E_0 \uplus B_0 \uplus B_1} r\rho$; also:

$$r\rho =_{E_0 \uplus B_0 \uplus B_1} r\theta|_{vars(l)}\rho = r\theta\rho = u\rho.$$

Hence $u' =_{E_0 \uplus B_0 \uplus B_1} u\rho \in [\![u]\!]_{\varphi'}$ by part (i).

□

Although the above soundness and completeness theorems, plus Lemma 3, show that $\rightarrow_{\mathcal{R}}$ is characterized symbolically by $\rightsquigarrow_{\mathcal{R}^\circ}$, for any rewrite theory $\mathcal{R}$ modulo $\mathcal{E}_0$, the relation $\rightsquigarrow_{\mathcal{R}^\circ}$ is in general undecidable because of Condition (c) in Definition 6. However, $\rightsquigarrow_{\mathcal{R}^\circ}$ becomes decidable for built-in theories $\mathcal{E}_0$ that can be extended to a *decidable theory* $\mathcal{E}_0^+$ (typically by adding some inductive consequences and the order on natural numbers) such that

$$(\forall \phi \in QF_{\Sigma_0}(X_0))\ \phi \text{ is } \mathcal{E}_0^+\text{-satisfiable} \iff (\exists \sigma : X_0 \longrightarrow T_{\Sigma_0})\ \mathcal{T}_{\mathcal{E}_0} \models \phi\sigma. \qquad (1)$$

Many decidable theories $\mathcal{E}_0^+$ of interest are supported by SMT solvers satisfying this requirement. For example, $\mathcal{E}_0$ can be the equational theory of natural number addition and $\mathcal{E}_0^+$ Pressburger arithmetic. That is, $\mathcal{T}_{\mathcal{E}_0}$ is the *standard model* of both $\mathcal{E}_0$ and $\mathcal{E}_0^+$, and $\mathcal{E}_0^+$-satisfiability coincides with satisfiability in such a standard model. Under such conditions, satisfiability of $\varphi \wedge \phi\zeta\theta$ (and therefore of φ') in a step $\langle t\,;\varphi\rangle \rightsquigarrow_{\mathcal{R}^\circ} \langle u\,;\varphi'\rangle$ becomes decidable by invoking an SMT-solver for $\mathcal{E}_0$, so that $\rightsquigarrow_{\mathcal{R}^\circ}$ can be naturally described as *symbolic rewriting modulo SMT* (and modulo B_1).

The symbolic reachability problems considered for a rewrite theory $\mathcal{R}$ modulo $\mathcal{E}_0$ in this paper, are existential formulas of the form $(\exists \overrightarrow{z})\ t \rightarrow^* u \wedge \varphi$, with $\overrightarrow{z}$ the variables appearing in t, u, and φ, $t, u \in T_\Sigma(X_0)_{State}$, and $\varphi \in QF_{\Sigma_0}(X_0)$. By abstracting the Σ_0-subterms of u, the ground solutions of such a reachability problem are those witnessing the model-theoretic satisfaction relation

$$\mathcal{T}_{\mathcal{R}} \models (\exists \overrightarrow{x} \uplus \overrightarrow{y})\ t(\overrightarrow{x}) \rightarrow^* u^\circ(\overrightarrow{y}) \wedge \varphi_1(\overrightarrow{x}) \wedge \varphi_2(\overrightarrow{x}, \overrightarrow{y}), \tag{2}$$

where $\mathcal{T}_{\mathcal{R}} = (\mathcal{T}_{\Sigma/E}, \rightarrow^*_{\mathcal{R}})$ is the initial reachability model of $\mathcal{R}$ [12], $t \in T_\Sigma(X_0)$ and $u^\circ \in T_{\Sigma_1}(X)$ are S_0-linear, $vars(t) \subseteq \overrightarrow{x} \subseteq X_0$, and $\overrightarrow{y} \subseteq X$. Thanks to the soundness and completeness results, Theorem 1, and Theorem 2, the solvability of Condition (b) for $\rightarrow_{\mathcal{R}}$ can be achieved by reachability analysis with $\rightsquigarrow_{\mathcal{R}^\circ}$, as stated in Theorem 3.

Theorem 3 (Symbolic Reachability Analysis)**.** *Let $\mathcal{R} = (\Sigma, E, R)$ be a rewrite theory modulo built-ins $\mathcal{E}_0$. The model-theoretic satisfaction relation in (2) has a solution iff there exist a term $v \in T_\Sigma(X)_{State}$, a constraint $\varphi' \in QF_{\Sigma_0}(X_0)$, and a substitution $\theta : X \longrightarrow T_\Sigma(X)$, with $dom(\theta) \subseteq \overrightarrow{y}$, such that (a) $\langle t\,;\varphi_1\rangle \rightsquigarrow^*_{\mathcal{R}^\circ} \langle v\,;\varphi'\rangle$, (b) $v =_{B_1} u^\circ\theta$, and (c) $\varphi' \wedge \varphi_2\theta$ is $\mathcal{T}_{\mathcal{E}_0}$-satisfiable.*

Proof. By theorems 1 and 2, and induction on the length of the rewrite derivation. □

In Theorem 3, since $dom(\theta) \subseteq \overrightarrow{y}$, and $\overrightarrow{x}$ and $\overrightarrow{y}$ are disjoint, the variables of $\overrightarrow{x}$ in $\varphi_2\theta$ are left unchanged. Therefore, $\varphi_2\theta$ *links* the requirements for the variables $\overrightarrow{x}$ in the initial state and $\overrightarrow{y}$ in the final state according to both φ_1 and φ_2. Also note that the inclusion of formula φ_1 as a conjunct in the formula in Condition (c) of Theorem 3 is superfluous because $\langle t\,;\varphi_1\rangle \rightsquigarrow_{\mathcal{R}^\circ} \langle v\,;\varphi'\rangle$ implies that φ_1 is a semantic consequence of φ'.

5. Reflective Implementation of $\rightsquigarrow_{\mathcal{R}^\circ}$

This section discusses the design and implementation of a prototype that offers support for symbolic rewriting modulo SMT in the Maude system. The prototype relies on Maude's meta-level features, that implement rewriting logic's reflective capabilities, and on SMT solving for $\mathcal{E}_0^+$ integrated in Maude as CVC3's decision procedures. The extension of Maude with CVC3 is available from the Matching Logic Project [44]. In the rest of this section, $\mathcal{R} = (\Sigma, E_0 \uplus B_0 \uplus B_1, R)$ is a rewrite theory modulo built-ins $\mathcal{E}_0$, where $\mathcal{E}_0$ satisfies Condition (1) in Section 4. The theory mapping $\mathcal{R} \mapsto \mathbf{u}(\mathcal{R})$ makes the rules unconditional by removing the constraints ϕ in the conditions of the rules in R.

In Maude, reflection is efficiently supported by its *META-LEVEL* module [14], which provides key functionality for rewriting logic's *universal theory* $\mathcal{U}$ [15]. In particular, rewrite theories $\mathcal{R}$ are meta-represented in $\mathcal{U}$ as terms $\overline{\mathcal{R}}$ of sort *Module*, and a term t in $\mathcal{R}$ is meta-represented in $\mathcal{U}$ as a term $\bar{t}$ of sort *Term*. The key idea of the reflective implementation is to reduce symbolic rewriting with $\rightsquigarrow_{\mathcal{R}^\circ}$ to *standard rewriting* in an associated reflective rewrite theory that extends the universal theory $\mathcal{U}$. This reduction is specially important for formal analysis purposes, because it makes available to $\rightsquigarrow_{\mathcal{R}^\circ}$ some formal analysis features provided by Maude for rewrite theories such as reachability analysis by search. This is illustrated by the case studies in sections 6 and 7.

The prototype defines a parametrized functional module $SAT(\Sigma_0, E_0 \uplus B_0)$ of quantifier-free formulas with Σ_0-equations as atoms. In particular, this module extends $(\Sigma_0, E_0 \uplus B_0)$ with new sorts *Atom* and *QFFormula*, and new *constants* $var(X_0)$ representing the variables X_0. It has, among other functions, a function $sat : QFFormula \longrightarrow Bool$ such that for ϕ, $sat(\phi) = \top$ if ϕ is $\mathcal{E}_0^+$-satisfiable, and $sat(\phi) = \bot$ otherwise.

The process of computing the one-step rewrites of a given constrained term $\langle t\,; \varphi\rangle$ under $\rightsquigarrow_{\mathcal{R}^\circ}$ is decomposed into two conceptual steps using Maude's metalevel. First, all possible triples $\langle u\,; \theta\,; \phi\rangle$ such that $t \rightarrow_{\mathbf{u}(\mathcal{R}^\circ)} u$ is witnessed by a matching substitution θ and a rule with constraint ϕ are computed[1]. Second, these triples are filtered out by keeping only those for which the quantifier-free formula $\varphi \wedge \phi\theta$ is $\mathcal{E}_0^+$-satisfiable.

The first step in the process is mechanized by function *next*, available from the parametrized module $NEXT(\overline{\mathcal{R}}, \overline{State}, \overline{QFFormula})$ where $\overline{\mathcal{R}}$, $\overline{State}$, and $\overline{QFFormula}$ are the metalevel representations, respectively, of the rewrite theory module $\mathcal{R}$, the state sort *State*, and the quantifier-free formula sort *QFFormula*. Function *next* uses Maude's *meta-match* function and the auxiliary function *new-vars* for computing fresh variables (see Section 4). In particular, the call $next(\overline{((S, \leq, F \uplus var(X_0)), E_0 \uplus B_0 \uplus B_1, R^\circ)}, \bar{t}, \overline{\varphi})$ computes all possible triples $\langle \overline{u}\,; \overline{\theta'}\,; \overline{\phi'}\rangle$ such that $t \rightsquigarrow_{\mathcal{R}^\circ} u$ is witnessed by a substitution θ' and a rule with constraint ϕ'. More precisely, such a call first computes a renaming $\zeta = fresh\text{-}vars(vars(t, \varphi))$ and then, for each rule$(l^\circ \rightarrow r$ **if** $\phi)$, it uses the function *meta-match* to obtain a substitution $\overline{\theta} \in meta\text{-}match(\overline{((S, \leq, F \uplus var(X_0)), B_0 \uplus B_1)}, \overline{t\downarrow_{E_0/B_0 \uplus B_1}}, \overline{l^\circ \zeta})$, and returns $\langle \overline{u}\,; \overline{\theta'}\,; \overline{\phi'}\rangle$ with $\overline{u} = \overline{r\zeta\theta}$, $\overline{\theta'} = \overline{\zeta\theta}$, and $\overline{\phi'} = \overline{\phi\zeta\theta}$. Note that by having a *deterministic* choice of fresh variables (including those in the constraint), function *next* is actually a *deterministic* function.

Using the above-mentioned infrastructure, the parametrized module *NEXT* implements the symbolic rewrite relation $\rightsquigarrow_{\mathcal{R}^\circ}$ as a *standard rewrite relation*, extending *META-LEVEL*, by means of the following conditional rewrite rule:

$$\begin{array}{ll} \mathbf{ceq} & \langle X{:}State\,; \varphi{:}QFFormula\rangle \rightarrow \langle Y{:}State\,; \varphi'{:}QFFormula\rangle \\ \mathbf{if} & \langle \overline{Y}\,; \overline{\theta}\,; \overline{\phi}\rangle\ S := next(\overline{\mathcal{R}^\bullet}, \overline{X}, \overline{\varphi}) \wedge sat(\varphi \wedge \phi) = \top \wedge \varphi' := \varphi \wedge \phi \end{array}$$

where $\mathcal{R}^\bullet = ((S, \leq, F \uplus var(X_0)), B, R^\circ)$. Therefore, a call to an external SMT solver is just an invocation of the function *sat* in $SAT(\Sigma_0, E_0 \uplus B_0)$ in order to achieve the above

[1]Note that in $\mathbf{u}(\mathcal{R}^\circ)$ variables in X_0 are interpreted as *constants*. Therefore, the number of matching substitutions θ thus obtained is finite.

functionality more efficiently and in a built-in way.

Given that the symbolic rewrite relation $\rightsquigarrow_{\mathcal{R}^\circ}$ is encoded as a standard rewrite relation, symbolic search can be *directly implemented* in Maude by its *search* command. In particular, for terms t, u°, constraints φ_1, φ_2, F a variable of sort *QFFormula*, the following invocation solves the inductive reachability problem in Condition (2):

$$search\ \ \langle t\,; \varphi_1\rangle \rightarrow^* \langle u^\circ\,; F\rangle\ \ such\ that\ \ sat(F \wedge \varphi_2).$$

6. Analysis of the CASH algorithm

This section presents an example, developed jointly with Kyungmin Bae, of a real-time system that can be symbolically analyzed in the prototype tool described in Section 5. The analysis applies model checking based on *rewriting modulo SMT*. Some details are omitted. Full details and the prototype tool can be found in [9].

The example involves the symbolic analysis of the CASH scheduling algorithm [13], which attempts to maximize system performance while guaranteeing that critical tasks are executed in a timely manner. This is achieved by maintaining a queue of unused execution budgets that can be reused by other jobs to maximize processor utilization. CASH poses non-trivial modeling and analysis challenges because it contains an unbounded queue. Unbounded data types cannot be modeled in timed-automata formalisms, such as those of UPPAAL [27] or Kronos [48], which assume a finite discrete state.

The CASH algorithm was specified and analyzed in Real-Time Maude by *explicit-state model checking* in an earlier paper by Ölveczky and Caccamo [36], which showed that, under certain variations on both the assumptions and the design of the protocol, it could miss deadlines. Explicit-state model checking has intrinsic limitations which the new analysis by rewriting modulo SMT presented below overcomes. The CASH algorithm is parametrized by: (i) the number N of servers in the system, and (ii) the values of a maximum budget b_i and period p_i, for each server $1 \leq i \leq N$. Even if N is fixed, *there are infinitely many initial states* for N servers, since the maximum budgets b_i and periods p_i range over the natural numbers. Therefore, explicit state model checking cannot perform a full analysis. If a counterexample for N servers exists, it may be found by explicit-state model checking for some chosen initial states, as done in [37], but it could be missed if the wrong initial states are chosen.

Rewriting modulo SMT is useful for symbolically analyzing infinite-state systems like CASH. Infinite sets of states are symbolically described by terms which may involve user-definable data structures such as queues, but whose only variables range over decidable types for which an SMT solving procedure is available. For the CASH algorithm, the built-in data types used are the Booleans (sort `iBool`) and the integers (sort `iInt`). Integer built-in terms are used to model discrete time. Boolean built-in terms are used to impose constraints on integers.

A symbolic state is a pair `{iB,Cnf}` of sort `Sys` consisting of a Boolean constraint `iB`, with *and* denoted `^`, and a multiset configuration of objects `Cnf`, with multiset union denoted by juxtaposition, where each object is a record like-structure with an object identifier, a class name, and a set of attribute-value pairs. In each object configuration there is a global object (of class `global`) that models the time of the system (with attribute name `time`), the priority queue (with attribute name `cq`), the availability (with attribute name `available`), and a deadline missed flag (with attribute name `deadline-miss`). A configuration can also contain any number of server objects (of class `server`). Each server object models the maximum budget (the maximum time within which a given job will be finished, with attribute name `maxBudget`), period (with attribute name `period`), internal state (with attribute name `state`), time executed (with attribute name `timeExecuted`), budget time used (with attribute name `usedOfBudget`), and time to deadline (with attribute name `timeToDeadline`). The symbolic transitions of CASH are specified by 14 conditional rewrite rules whose conditions specify constraints solvable by the SMT decision procedure. For example, rule `[deadlineMiss]` below models the detection of a deadline miss for a server with non-zero maximum budget.

```
vars AtSG AtS : AttributeSet .  var iB : iBool .         var Cnf : Configuration .
vars iT iT' iNZT : iInt .       var St : ServerState .   vars G S : Oid .  var B : Bool .

crl [deadlineMiss] :
    { iB, < G : global | dead-miss |-> B, AtSG >
          < S : server | state |-> St, usedOfBudget |-> iT, timeToDeadline |-> iT',
                         maxBudget |-> iNZT, AtS > Cnf }
=> {iB ^ iT >= c(0) ^ iNZT > c(0) ^ iT' >  c(0) ^ iNZT > iT + iT',
          < G : global | dead-miss |-> true, AtSG >
          < S : server | state |-> St, usedOfBudget |-> iT, timeToDeadline |-> iT',
                         maxBudget |-> iNZT, AtS > Cnf }
if St =/= idle /\ check-sat(iB ^ iT >= c(0) ^ iNZT > c(0) ^ iT' >  c(0) ^ iNZT > iT + iT') .
```

That is, the protocol misses a deadline for server `S` whenever the value of attribute `maxBudget` exceeds the addition of values for `usedOfBudget` and `timeToDeadline` (i.e., `iNZT > iT + iT'`) , so that the allocated execution time cannot be exhausted before the server's deadline.

The goal is to verify *symbolically* the existence of missed deadlines of the CASH algorithm for the *infinite set of initial configurations* containing two server objects s_0 and s_1 with maximum budgets b_0 and b_1 and periods p_0 and p_1 as unspecified natural numbers, and such that each server's maximum budget is strictly smaller than its period (i.e., $0 \leq b_0 < p_0 \wedge 0 \leq b_1 < p_1$). This infinite set of initial states is specified symbolically by the equational definition (not shown) of term `symbinit`. Maude's `search` command can then be used to symbolically check if there is a reachable state for any ground instance of `symbinit` that misses the deadline:

```
search in SYMBOLIC-FAILURE : symbinit =>*
  { iB:iBool,  Cnf:Configuration < g : global | AtS:AttributeSet, deadline-miss |-> true > } .
Solution 1 (state 233)
states: 234  rewrites: 60517 in 2865ms cpu (2865ms real) (21118 rewrites/second)
iB:iBool --> ((i(0) <= c(0) ^ i(1) <= c(0)) v i(0) <= c(0) + i(1) ^ ...
Cnf:Configuration -->
< s1 : server | maxBudget |-> i(0), period |-> i(1), state |-> waiting, usedOfBudget |-> c(0),
                timeToDeadline |-> ((i(1) -- c(1)) -- c(1)), timeExecuted |-> c(0) >
< s2 : server | maxBudget |-> i(2), period |-> i(3), state |-> executing, usedOfBudget |-> c(2),
                timeToDeadline |-> ((i(3) -- c(1)) -- c(1)), timeExecuted |-> c(2) >
AtS:AttributeSet --> time |-> c(2), cq |-> emptyQueue, available |-> false
```

A counterexample is found at (modeling) time two, after exploring 233 symbolic states in less than 3 seconds. By using a satisfiability witness of the constraint `iB` computed by the search command, a concrete counterexample is found by exploring only 54 ground states. This result compares favorably, in both time and computational resources, with the ground counterexample found by explicit-state model checking in [36], where more that 52,000 concrete states were explored before finding a counterexample.

7. Symbolic Reachability Analysis for PLEXIL Modulo Integer Constraints

Synchronous languages were introduced in the 1980s to program *reactive systems*, i.e., open systems whose behavior is determined by their continuous reaction to the environment where they are deployed. The *Plan Execution Interchange Language* (PLEXIL) [20] is a synchronous language developed by NASA to support autonomous spacecraft operations. Given the safety-critical nature of spacecraft operations, PLEXIL's operational semantics has been formally defined [17] and several properties of the language, such as determinism and compositionality, have been mechanically verified [16] in the Prototype Verification System (PVS) [38]. A rewriting logic semantics of PLEXIL [18] has been developed in Maude and has been used, within a formal interactive verification environment [41], to validate the intended semantics of the language against a wide variety of plan examples.

PLEXIL programs define reactive systems that interact with an external environment of sensors and actuators. Such programs are *deterministic* by assuming a given concrete value for each of the sensors that the reactive system interacts with. Therefore, to execute by standard rewriting the rewriting logic semantics in [18] (and perform various kinds of reachability analysis verification in Maude using such rewriting), *concrete values of the data in sensors* had to be assumed for the reactive interactions. Since, in general, the possible tuples of such values can be infinite or (assuming finite arithmetic precision) extremely large, the concrete executions and formal analyses allowed by the concrete rewriting semantics had to be necessarily incomplete. This is analogous to the incompleteness of simulating and analyzing the CASH algorithm example in Real-Time Maude, versus the complete analysis by rewriting modulo SMT presented in Section 6. Using rewriting modulo SMT, a *complete* rewriting logic semantics that can symbolically cover all possible values in an external environment has been defined for PLEXIL in [40].

This section presents a case study overview on the symbolic analysis of reachability properties for a large subset of the PLEXIL language based on rewriting modulo SMT, which extends and complements the rewriting logic semantics of the language. Such an analysis is able to automatically detect reachability violations on input plans where the values of external variables can be left unspecified, a task that is impossible to achieve with the *ground* rewriting logic semantics of the language.

7.1. PLEXIL Overview

This section presents an overview of PLEXIL; the reader is referred to [20] for a detailed description of the language.

A PLEXIL program, called a *plan*, is a tree of *nodes* representing a hierarchical decomposition of tasks. Interior nodes, called *list nodes*, provide control structure and naming scope for local variables. The primitive actions of a plan are specified in the leaf nodes. Leaf nodes can be *assignment nodes*, which assign values to local variables, *command nodes*, which call external commands, or *empty nodes*, which do nothing. PLEXIL plans interact with a functional layer that provides the interface with the external environment. This functional layer executes the external commands and communicates the status and result of their execution to the plan through *external variables*.

Nodes have an *execution state*, which can be *inactive*, *waiting*, *executing*, *iterationend*, *failing*, *finishing*, or *finished*, and an *execution outcome*, which can be *unknown*, *skipped*, *success*, or *failure*. They can declare local variables that are accessible to the node in which they are declared and all its descendants. In contrast to local variables, the execution state and outcome of a node are visible to all nodes in the plan. Assignment nodes also have a *priority*, which is used to solve race conditions. The *internal state* of a node consists of the current values of its execution state, execution outcome, and local variables.

Each node is equipped with a set of *gate conditions* and *check conditions* that govern the execution of a plan. Gate conditions provide control flow mechanisms that react to external events. In particular, the *start condition* specifies when a node starts its execution, the *end condition* specifies when a node ends its execution, the *repeat condition* specifies when a node can repeat its execution, and the *skip condition* specifies when the execution of a node can be skipped. Check conditions are used to signal abnormal execution states of a node and they can be either *pre-condition*, *post-condition*, or *invariant* conditions. The language includes Boolean, integer and floating-point arithmetic, and string expressions. It also includes *lookup expressions* that read the value of external variables provided to the plan through the executive. Expressions appear in conditions, assignments, and arguments of commands. Each of the basic types is extended by a special value *unknown* that can result, for example, when a lookup fails.

The execution of a plan in PLEXIL is driven by external events from the environment that trigger changes in the gate conditions. All nodes affected by a change in a gate condition synchronously respond to the event by modifying their internal state. These internal modifications may trigger more changes in gate conditions that in turn are synchronously processed until quiescence is reached for all nodes involved. External events are considered in the order in which they are received. An external event and all its cascading effects are processed before the next event is considered. This behavior is known as *run-to-completion semantics*.

The *atomic relation* describes the execution of an individual node in terms of state transitions triggered by changes in the environment. The *micro relation* describes the *synchronous* reduction of the atomic relation with respect to the *maximal redexes strategy*, i.e., the synchronous application of the atomic relation to the maximal set of nodes of a plan. The remaining three relations are the *quiescence relation*, the *macro relation*, and the *execution relation* that describe, respectively, the reduction of the micro relation until normalization, the interaction of a plan with the external environment upon one external event, and the *n*-iteration of the macro relation corresponding to *n* time steps. Figure 1 depicts the transition diagram defining PLEXIL's atomic transitions for lists in state *executing*.

Since local variables declared in a node are shared by its children nodes, it may be possible that two nodes attempt to synchronously write the same variable. The priority

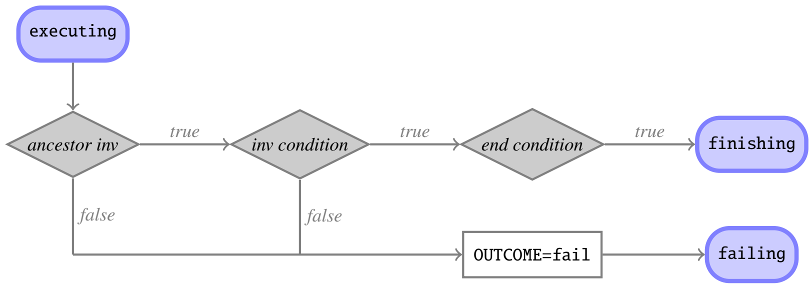

Figure 1: Atomic transitions for list nodes in state *executing*.

```
AssignWithConflict: {
  Integer       x = 0;
  Invariant:    x >= 0;
  NodeList:
  NonNeg: {
    Start:      Lookup(S) >= 0;
    Assignment: x := 1;
  }
  NonPos: {
    Start:      Lookup(S) <= 0;
    Assignment: x := 2;
  }
}
```

Figure 2: A parallel assignment with a potential race condition.

mechanism included in the semantics of PLEXIL can be used by programmers to deal with this problem. Unfortunately, priorities are optional and, in practice, race conditions may occur during the execution of a PLEXIL program. For instance, consider the plan `AssignWithConflig` in Figure 2. This plan has one list node and two assignment nodes, `NonNeg` and `NonPos`. It declares a local integer memory `x` and interacts with the external environment via the integer variable `S`. Note that depending on the value of `S`, the assignment nodes `NonNeg` and `NonPos` may or may not start execution, and a race condition can happen on `x` when the value of `S` is 0. With the symbolic semantics presented in this section, the race condition on `x` can be automatically detected.

7.2. Symbolic Detection of Race Conditions

Detection of race conditions on local memories and violation of node invariants are important in PLEXIL. As such, predicates for checking them are already available from the symbolic semantics. In particular, states predicates `inv` and `race-free`, which take an argument of sort `NeQualified` (i.e., the sort of node identifiers) are offered to the user.

The intended semantics of the state predicates is with respect to the initial semantics of PLEXIL. For example, consider the following definition of `inv` in the syntax of Maude model checker:

```
eq ({ iB:Bool,
      < O:NeQualified : C:Cid | inv |-> iB':iBool, AtS:AttributeSet >
      Cnf:Configuration }) |= inv(O:NeQualified)
 = check-unsat(iB:iBool and
             not(eval(< O:NeQualified : C:Cid |
                         inv|-> iB':iBool, AtS:AttributeSet > Cnf,
                    iB':iBool))) .
```

The invariant condition of node `O` represented by the Boolean expression `iB'` yields an invariant violation for `O` whenever the conjunction of the state's constraint `iB` and the negation of `iB'` is unsatisfiable. This precisely means that there is a ground counter-example state for the invariance of the node.

Boolean and integer expressions can be evaluated 'symbolically' by means of function `eval`, while function `check-unsat` implements the call to CVC3 :

```
op eval : Configuration iBool -> iBool .
op eval : Configuration iInt -> iInt .
op check-unsat : iBool -> Bool .
```

The evaluation of an expression by `eval` is given w.r.t. an object configuration and it is equationally defined recursively on the complexity of expressions.

Recall the plan `AssignWithConflict` in Figure 2, which has a potential race condition for the local memory `x`. Assume that `SPLX` represents the symbolic rewriting logic semantics of PLEXIL, and let `init` be a configuration of objects representing an initial configuration for `AssignWithConflict`. Consider the following safety verification requirements:

$$\mathcal{T}_{\text{SPLX}}, \{\texttt{c(true)}, \texttt{init}\} \models \Box\texttt{race-free(x.AssignWithConflict)}, \tag{3}$$

$$\mathcal{T}_{\text{SPLX}}, \{\texttt{i(0) >= c(1)}, \texttt{init}\} \models \Box\texttt{race-free(x.AssignWithConflict)}, \tag{4}$$

$$\mathcal{T}_{\text{SPLX}}, \{\texttt{i(0) >= c(1)}, \texttt{init}\} \models \Box\texttt{inv(AssignWithConflict)}. \tag{5}$$

The external variable `S` in `AssignWithConflict` is represented by the Boolean term `i(0)`. Property (3) states that there is no race condition on memory `x` if `i(0)` has no initial constraints. Property (4) states that there is no race condition on memory `x` if `i(0)` is assumed to be at least 1. Property (5) states that the invariant condition of node `AssignWithConflict` holds if `i(0)` is assumed to be at least 1. Note that these properties are symbolic reachability requirements because of the nature of the external variable S. Also, the constrained terms defining the initial states in these properties represent, in each case, infinitely many initial states.

By directly using Maude's LTL Model Checker, property (3) can be disproved, and properties (4) and (5) can be proved automatically.

```
==========================================
reduce in ASSIGNWITHCONFLICT :
  verify-lite({c(true), init}, [] race-free(x . AssignWithConflict)) .
rewrites: 2590 in 525ms cpu (1629ms real) (4929 rewrites/second)
result Bool: false
==========================================
reduce in ASSIGNWITHCONFLICT :
  verify-lite( { i(0) >= c(1), init}, [] race-free(x . AssignWithConflict)) .
```

```
rewrites: 2846 in 575ms cpu (614ms real) (4947 rewrites/second)
result Bool: true
==========================================
reduce in ASSIGNWITHCONFLICT :
  verify-lite( {i(0) >= c(1), init}, [] inv(AssignWithConflict) .
rewrites: 3191 in 576ms cpu (702ms real) (5534 rewrites/second)
result Bool: true
```

Function `verify-lite` is a wrapper to Maude's LTL Model Checker function `modelCheck`. This mapping outputs either `true` or `false`, depending on the output of the model checker function, omitting a counterexample if any.

8. Related Work and Concluding Remarks

The idea of combining term rewriting/narrowing techniques and constrained data structures is an active area of research, specially since the advent of modern theorem provers with highly efficient decision procedures in the form of SMT solvers. The overall aim of these techniques is to advance applicability of methods in symbolic verification where the constraints are expressed in some logic that has an efficient decision procedure. In particular, the work presented here has strong similarities with the narrowing-based symbolic analysis of rewrite theories initiated in [31] and extended in [8]. The main difference is the replacement of narrowing by SMT solving and the decidability advantages of SMT for constraint solving.

M. Ayala-Rincón [5] investigates, in the setting of many-sorted equational logic, the expressiveness of conditional equational systems whose conditions may use built-in predicates. This class of equational theories is important because the combination of equational and built-in premises yield a type of clauses which is more expressive than purely conditional equations. Rewriting notions like confluence, termination, and critical pairs are also investigated. S. Falke and D. Kapur [21] studied the problem of termination of rewriting with constrained built-ins. In particular, they extended the dependency pairs framework to handle termination of equational specifications with semantic data structures and evaluation strategies in the Maude functional sublanguage. The same authors used the idea of combining rewriting induction and linear arithmetic over constrained terms [22]. Their aim is to obtain equational decision procedures that can handle semantic data types represented by the constrained built-ins. H. Kirchner and C. Ringeissen proposed the notion of constrained rewriting and have used it by combining symbolic constraint solvers [25]. The main difference between their work and rewriting modulo SMT presented in this paper is that the former uses narrowing for symbolic execution, both at the symbolic 'pattern matching' and the constraint solving levels. In contrast, rewriting modulo SMT solves the symbolic pattern matching task by rewriting while constraint solving is delegated to an SMT decision procedure. More recently, C. Kop and N. Nishida [26] have proposed a way to unify the ideas regarding equational rewriting with logical constraints. More generally, while the approaches in [5, 21, 22, 25, 26] address symbolic reasoning for *equational* theorem proving purposes, none of them addresses the kind of non-deterministic rewrite rules, which are needed for open system modeling. More recently, A. Arusoaie et al. [4] have proposed a language-independent symbolic execution framework, within the *K* framework [28], for languages endowed with a formal operational semantics based on term rewriting.

There, the built-in subtheories are the datatypes of a programming language and symbolic analysis is performed on constrained terms (called "patterns"); unification is also implemented by matching for a restricted class of rewrite rules and uses SMT solvers to check constraints.

This paper has presented rewrite theories modulo built-ins and has shown how they can be used for *symbolically* modeling and analyzing concurrent open systems, where non-deterministic values from the environment can be represented by built-in terms [40, 42]. In particular, the main contributions of this paper can be summarized as follows: (1) it presents rewriting modulo SMT as a new symbolic technique combining the powers of rewriting, SMT solving, and model checking; (2) this combined power can be applied to model and analyze systems outside the scope of each individual technique; (3) in particular, it is ideally suited to model and analyze the challenging case of *open systems*; and (4) because of its reflective reduction to standard rewriting, current algorithms and tools for model checking closed systems can be *reused* in this new symbolic setting without requiring any changes to their implementation.

Under reasonable assumptions, including decidability of $\mathcal{E}_0^+$, a rewrite theory modulo is executable by term rewriting modulo SMT. This feature makes it possible to use, for symbolic analysis, state-of-the-art tools already available for Maude, such as its space search commands, with no change whatsoever required to use such tools. We have proved that the symbolic rewrite relation is sound and complete with respect to its ground counterpart, have presented an overview of the prototype that offers support for rewriting modulo SMT in Maude, and have presented two case studies on the symbolic analysis of the CASH scheduling algorithm and the PLEXIL synchronous language illustrating the use of these techniques.

Future work on a mature implementation and on extending the idea of rewriting modulo SMT with other symbolic constraint solving techniques such as narrowing modulo should be pursued. Also, the extension to other symbolic LTL model checking properties, together with state space reduction techniques, should be investigated. Further applications to Real-Time Maude, PLEXIL, and other languages should also be pursued.

Acknowledgments. This work was partially supported by NSF Grant CNS 13-19109. The first author would like to thank the National Institute of Aerospace for a short visit supported by the Assurance of Flight Critical System's project of NASA's Aviation Safety Program at Langley Research Center under Research Cooperative Agreement No. NNL09AA00A.

References

References

[1] E. Althaus, E. Kruglov, and C. Weidenbach. Superposition modulo linear arithmetic SUP(LA). In S. Ghilardi and R. Sebastiani, editors, *FroCos*, volume 5749 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2009.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. *Model Checking Software*, pages 146–162, 2006.

[4] A. Arusoaie, D. Lucanu, and V. Rusu. A generic framework for symbolic execution. In M. Erwig, R. F. Paige, and E. V. Wyk, editors, *Software Language Engineering - 6th International Conference, SLE 2013, Indianapolis, IN, USA, October 26-28, 2013. Proceedings*, volume 8225 of *Lecture Notes in Computer Science*, pages 281–301. Springer, 2013.

[5] M. Ayala-Rincón. *Expressiveness of Conditional Equational Systems with Built-in Predicates*. PhD thesis, Universität Kaiserslauten, 1993.

[6] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[7] F. Baader and K. Schulz. Unification in the union of disjoint equational theories: combining decision procedures. *Journal of Symbolic Computation*, 21:211–243, 1996.

[8] K. Bae, S. Escobar, and J. Meseguer. Abstract logical model checking of infinite-state systems using narrowing. In F. van Raamsdonk, editor, *RTA*, volume 21 of *LIPIcs*, pages 81–96. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[9] K. Bae and C. Rocha. A note on symbolic reachability analysis modulo integer constraints for the CASH algorithm. Available at: `http://maude.cs.uiuc.edu/cases/scash`, 2012.

[10] M. P. Bonacina, C. Lynch, and L. M. de Moura. On deciding satisfiability by theorem proving with speculative inferences. *Journal of Automated Reasoning*, 47(2):161–189, 2011.

[11] A. Boudet. Combining unification algorithms. *J. Symb. Comp.*, 16(6):597–626, 1993.

[12] R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *Theoretical Computer Science*, 360(1-3):386–414, 2006.

[13] M. Caccamo, G. C. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *IEEE Real-Time Systems Symposium*, pages 295–304. IEEE Computer Society, 2000.

[14] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007.

[15] M. Clavel, J. Meseguer, and M. Palomino. Reflection in membership equational logic, many-sorted equational logic, horn logic with equality, and rewriting logic. *Theoretical Computer Science*, 373(1-2):70–91, 2007.

[16] G. Dowek, C. Muñoz, and C. Păsăreanu. A formal analysis framework for PLEXIL. In *Proceedings of 3rd Workshop on Planning and Plan Execution for Real-World Systems*, pages 45–51, September 2007.

[17] G. Dowek, C. Muñoz, and C. Păsăreanu. A small-step semantics of PLEXIL. Technical Report 2008-11, National Institute of Aerospace, Hampton, VA, 2008.

[18] G. Dowek, C. Muñoz, and C. Rocha. Rewriting logic semantics of a plan execution language. *CoRR*, abs/1002.2872, 2010.

[19] F. Durán and J. Meseguer. On the Church-Rosser and coherence properties of conditional order-sorted rewrite theories. *Journal of Logic and Algebraic Programming*, to appear, 2011.

[20] T. Estlin, A. Jónsson, C. Păsăreanu, R. Simmons, K. Tso, and V. Verma. Plan Execution Interchange Language (PLEXIL). Technical Memorandum TM-2006-213483, NASA, 2006.

[21] S. Falke and D. Kapur. Operational termination of conditional rewriting with built-in numbers and semantic data structures. *ENTCS*, 237:75–90, 2009.

[22] S. Falke and D. Kapur. Rewriting induction + linear arithmetic = decision procedure. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2012.

[23] M. Ganai and A. Gupta. Accelerating high-level bounded model checking. In *ICCAD*, pages 794–801. ACM, 2006.

[24] J. A. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.

[25] H. Kirchner and C. Ringeissen. Combining symbolic constraint solvers on algebraic domains. *Journal of Symbolic Computation*, 18(2):113–155, 1994.

[26] C. Kop and N. Nishida. Term rewriting with logical constraints. In *FroCos*, volume 8152 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2013.

[27] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *STTT*, 1(1-2):134–152, 1997.

[28] D. Lucanu, T.-F. Serbanuta, and G. Rosu. K framework distilled. In *Rewriting Logic and Its Applications - 9th International Workshop, WRLA 2012, Held as a Satellite Event of ETAPS, Tallinn, Estonia, March 24-25, 2012, Revised Selected Papers*, volume 7571 of *Lecture Notes in Computer Science*, pages 31–53. Springer, 2012.

[29] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[30] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *WADT*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1997.

[31] J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Higher-Order and Symbolic Computation*, 20(1-2):123–160, 2007.

[32] A. Milicevic and H. Kugler. Model checking using SMT and theory of lists. *NASA Formal Methods*, pages 282–297, 2011.

[33] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[34] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(t). *Journal of the ACM*, 53(6):937–977, 2006.

[35] E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer-Verlag, 2002.

[36] P. C. Ölveczky and M. Caccamo. Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude. In L. Baresi and R. Heckel, editors, *Fundamental Approaches to Software Engineering (FASE'06)*, volume 3922 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2006.

[37] P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symbolic Computation*, 20(1-2):161–196, 2007.

[38] S. Owre, J. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[39] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *LNAI*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[40] C. Rocha. *Symbolic Reachability Analysis for Rewrite Theories*. PhD thesis, University of Illinois at Urbana-Champaign, 2012.

[41] C. Rocha, H. Cadavid, C. A. Muñoz, and R. Siminiceanu. A formal interactive verification environment for the plan execution interchange language. In J. Derrick, S. Gnesi, D. Latella, and H. Treharne, editors, *IFM*, volume 7321 of *Lecture Notes in Computer Science*, pages 343–357. Springer, 2012.

[42] C. Rocha, J. Meseguer, and C. Muñoz. Rewriting modulo SMT. Technical Memorandum NASA/TM-2013-218033, NASA, Langley Research Center, Hampton VA 23681-2199, USA, August 2013.

[43] C. Rocha, J. Meseguer, and C. Muñoz. Rewriting modulo SMT and open system analysis. In S. Escobar, editor, *Rewriting Logic and Its Applications*, volume –To Appear– of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2014.

[44] G. Roşu and A. Ştefănescu. Matching Logic: A New Program Verification Approach (NIER Track). In *ICSE'11: Proceedings of the 30th International Conference on Software Engineering*, pages 868–871. ACM, 2011.

[45] M. Veanes, N. Bjørner, and A. Raschke. An SMT approach to bounded reachability analysis of model programs. In *FORTE*, pages 53–68. Springer, 2008.

[46] P. Viry. Equational rules for rewriting logic. *TCS*, 285:487–517, 2002.

[47] D. Walter, S. Little, and C. Myers. Bounded model checking of analog and mixed-signal circuits using an SMT solver. *Automated Technology for Verification and Analysis*, pages 66–81, 2007.

[48] S. Yovine. Kronos: A verification tool for real-time systems. *STTT*, 1(1-2):123–133, 1997.